LEARNING TO PLAN NEAR-OPTIMAL COLLISION-FREE PATHS

Alex W. Ho and Geoffrey C. Fox

Concurrent Computation Program
206-49, California Institute of Technology,
Pasadena, CA 91125, USA

Abstract

A new approach to find a near-optimal collision-free path is presented. The path planner is an implementation of the adaptive error back-propagation algorithm which learns to plan "good", if not optimal, collision-free paths from human-supervised training samples.

Path planning is formulated as a classification problem in which class labels are uniquely mapped onto the set of maneuverable actions of a robot or vehicle. A multi-scale representational scheme maps physical problem domains onto an arbitrarily chosen fixed size input layer of an error back-propagation network. The mapping does not only reduce the size of the computation domain, but also ensures applicability of a trained network over a wide range of problem sizes. Parallel implementation of the neural network path planner on hypercubes or Transputers based on Parasoft EXPRESS is simple and efficient. Simulation results of binary terrain navigation indicate that the planner performs effectively in unknown environment in the test cases.

Introduction

Robots have been successfully employed in very restricted, mechanical, and repetitive tasks such as to improve productivity and quality in assembly lines in automotive industry. Although it is not likely that man can construct even a near general-purpose robot in the foreseeable future given the current level of technology and advancement of science, the future generation of task-specific robot systems are expected to be more "autonomous" and "intelligent". These future robot systems would posses highly integrated capabilities of task-specific sensing – to gather relevant information of the environment and construct a limited world model of the physical surroundings, goal-oriented planning – to achieve a high level specification of a goal by generating a sequence of robot actions in advance, motor control – to execution the planned sequence of actions step by step, and learning – to gain domain-specific knowledge from experience and response to unknown environment intelligently. In this paper, we confined our study to path planning for robot navigation.

The objective of developing autonomous robot navigation controller is to enable a robot to guide itself moving from one point of space to a destination without collision with the obstacles in its environment. The most basic form of a motion planning problem is the generalized mover's problem, which is also known as the Findpath or obstacle avoidance problem [1.] The goal is to find *any* collision-free path. For economic reasons, the path that a robot tracks should obey some constraints, which is usually in time and/or energy usages. For all practical purposes, the notion of planning a "good" path is of prime importance to any reasonable navigation controller.

There are many variants of the path planning problem. The task of planning an optimal path is achievable only for simple problems. Most of the time, the amount of computation required to obtain such a path could be costly. In many circumstances, optimal paths are not required. It is often more important to obtain a "good" (i.e., nearly but not precisely optimal) path quickly than to devote precious computational resources to find the exact solution. In fact the input data is often imprecise (e.g. the exact nature of the terrain is not known) and the notion of a precise optimal path undefined. Several new optimization techniques such as simulated annealing, neural networks, elastic networks and genetic algorithms have been devised for such approximate optimization problems [2-7.]

0-8186-2113-3/90/0000/0131$01.00 © 1990 IEEE

The use of a multi-layer feedforward neural network for path planning was first reported in [8], and some preliminary results on performance of such a trained network were discussed in [9.] In this paper, we will describe in detail the implementation of the neural approach used in [8,9] to the problem of planning a near-optimal, collision-free path for a single mobile robot moving in 2-dimensional binary terrains. Computational performance of the parallel algorithm on several distributed-memory MIMD processors like NCUBE-1, MEIKO Computing Surface (Transputer-based system), and iPSC-2 are compared. We will present algorithmic and implementation performances for cases of robot navigation on binary terrain.

Modeling The Navigation Terrains

We have chosen to apply our new approach to the simplest non-trivial path planning problem which is the navigation of a single vehicle in a plane with binary terrain. The terrain partitioning is a standard grid tessellation of the physical space of the problem domain which contains regions of random or structured obstacles. The remaining regions are robot traversable space. The discretized binary problem domain is represented as a 2-d matrix. A measure of the size of an instance of the path planning problem is the number of elements in the 2-d matrix. The higher the resolution of discretization, the bigger the problem size.

Although our technique can be extended to cover a more general path planning problem, the problem statement of our current study is stated as:

> Given a discretized 2-d rectangular physical domain R, a distribution of binary obstacles D over R, the robot's current position $S \in R - D$, a high level specification of the target position $T \in R - D - S$, and a set of maneuverable actions or motor control constraints C governing the robot, find a near-optimal, collision-free path for the robot to move around in R from S to T.

Supervised Neural Network Approach

Artificial neural networks have been employed in a variety of applications, and were found most useful in the class of applications which a human operator can handle easily and efficiently. Obvious examples are in pattern classification and speech recognition. Another example is in playing chess. It has been a long-standing speculation that a good chess player recognizes abstracted patterns of the current board and commands a move with efficacy, while a chess playing computer program has to evaluate and search a huge game tree of legal moves and countermoves rooted from the current board, iteratively to a fixed depth.

In the same vein, the path planning problem can be transformed to a pattern classification problem in which class labels are uniquely mapped onto the set of maneuverable actions C of a robot, while each time instance of a scenario is mapped onto a 2-d binary pattern. We used a multi-layer perceptrons based on the adaptive error back-propagation algorithm [10] as the pattern classifier for the transformed path planning problem.

The back-propagation model has been widely used for pattern recognition tasks. The architecture of such a neural net model has an input, output, and intermediate layers. All inter-layers are fully connected. Unlike the Hopfield model which has recurrent connections[4,] the perceptrons model does not provide a feedback mechanism for neuronal activation to propagate. A set of training sample pairs which carries some form of relevant information about the classification problem at hand is used to train the multi-layer network. Iterative synaptic weight adaptation occurs following the back-propagation algorithm as the error signal at the output layer is propagated backward and filtered by the same set of synaptic connections for forward propagation.

The Issue of Representation

The success of the back-propagation model on pattern recognition problems relies heavily on the choice of the pre-processing operations. The choice of pre-processing operations for raw pattern data determine the selective pruning and encoding of information. The representations that emerge from these operations impose constraints on subsequent processing by the back-propagation neural model.

A reasonable pre-processing scheme should be one which reduces and encodes raw patterns into some form of standard representations. We adopted a non-linear, multi-scale sampling strategy which

mapped a physical problem domain onto an arbitrarily chosen fixed size input layer of an error back-propagation network. The multi-scale representation of patterns is a natural consequence of the sampling strategy used.

The sampling strategy involves using high resolution neurons to encode terrain information close to a robot, and progressively coarser neurons away from a robot by increasing the sampling interval. The physical problem domain is separated, in a fuzzy sense, into a near field and a far field. Near field information which is encoded in high resolution neurons is used to generate immediate action (corresponds to local planning,) while far field information which is encoded in the coarse neurons is used for global planning.

The only traffic regulation imposed on robot motion is that every move must be collision-free. In this study, a robot was restricted to move in one of the five admissible directions in a 2-d rectangular problem domain, and the cost to move in any of the five admissible directions is the same. Our formulation of robot maneuverable motions could be extended to eight directions to include the eight nearest neighbor in the case of a 2-d grid tessellation. Figure 1 shows the five admissible moves of a robot. Since the cost of selecting to move in any of these five directions are the same, an optimal collision-free path would be one which minimizes the number of moves required to get from a source point to a target position without violating the traffic regulation.

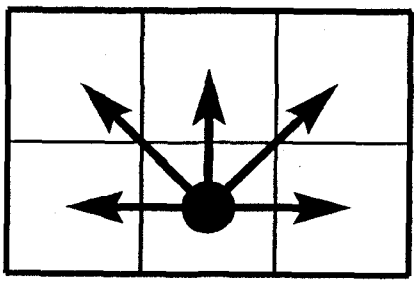

Figure 1: Vehicle is constrainted to move in one of the five directions. All moves have the same cost.

The five admissible moves are mapped one to one onto five grandmother cells at the output layer of a back-propagation model. The activation values of the five grandmother cells indicate how good it is to move a robot in each of the five corresponding directions (see Fig. 2.)

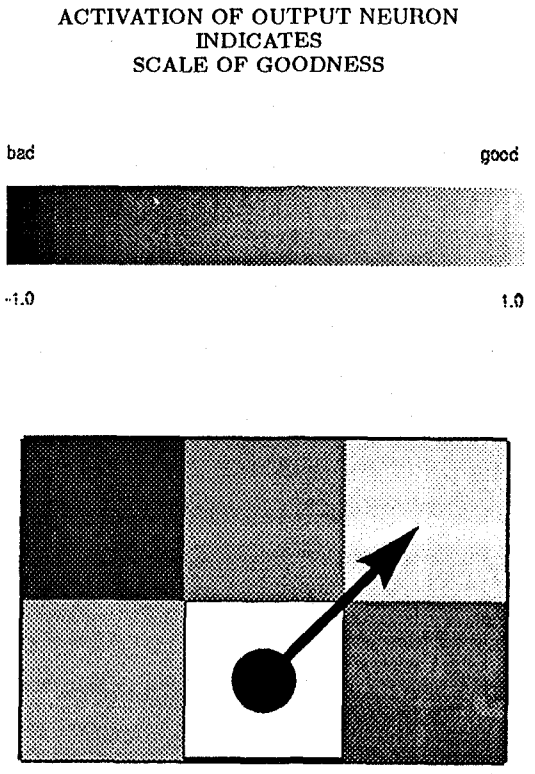

Figure 2: A robot moves in the direction which corresponds to the highest activation voltage.

We have arbitrarily chosen to use four different scales and nine general directions (East, NEE, NE, NNE, North, NNW, NW, NWW, West) to represent each scenario. This sampling strategy divides the problem domain into 36 regions. At the input layer, one neuron is needed to encode information for each region which is actually one combination of direction and scale. All together, 36 neurons are needed. An intermediate layer with 20 neurons was used. The number of neurons in the intermediate layer was arbitrarily chosen to achieve a fan-in architecture.

The activation value for each input neuron is a function of a porosity index which is a measure of the traversability of the corresponding region. The porosity index is taken as the compliment of the density of obstacles. The input neuron activation is computed by using Eq. (1)

$$V_{\eta_i} = f(1-\rho, \gamma_i) + \delta(target, \gamma_i) \qquad (1)$$

where $V_{\eta_i} \in [-1.0, 1.0]$ is the activation value for neuron η_i in the region γ_i, ρ is the density of obstacles in γ_i, and δ is the Kronecker delta which equals to 1 if the target position is in γ_i, and zero otherwise.

Parallel Back-Propagation

The back-propagation algorithm is an effective training algorithm for the feed-forward multi-layer perceptron model. It is a generalization of the least mean square algorithm or the delta rule. Back-propagation uses a gradient descent technique to minimize a quadratic error function which is defined as the mean square differences between the pair of actual network output vector and its associated target vector for the set of training samples.

Let us define the following:

- w_{ij} is the connection weight between the j^{th} neuron in the current layer and the i^{th} neuron in the immediate lower layer,
- θ_j is the internal threshold of the j^{th} neuron in the current layer,
- x_i is the i^{th} continuous-valued input from the input layer or the layer underneath the current layer,
- x'_j is the output of the j^{th} neuron in the current layer,
- y_j is the actual model output of the j^{th} neuron in the output layer,
- and d_j is the desired or target output of the j^{th} neuron in the output layer.

The model is trained by initially assigning small random weights to the synaptic connections and small random thresholds to the artificial neurons. The neuronal outputs from each layer are then computed by

$$x'_j = f\Big(\sum_{i=0,\ldots,N-1} w_{ij}x_i - \theta_j\Big) \qquad (2)$$

where N is the number of neurons in the layer below the current layer. If the current layer is the physical output layer,

$$y_j = x'_j. \qquad (3)$$

The neuron activation function f has to be non-decreasing, continuously differentiable. Usually, the sigmoid logistic function

$$f(\zeta) = \frac{1}{1 + e^{-(\zeta)}} \qquad (4)$$

is used for this purpose.

Given the measure of the error on any pattern in the training set as

$$E = \sum_j (d_j - y_j)^2 \qquad (5)$$

and the neuron activation function f as described in Eq. (4), adaptive correction of the connection weights in the direction of $-\partial E/\partial w$ corresponds to performing a steepest descent search in the weight space to minimize error. Synaptic weight adaptation follows Eq. (6)

$$\Delta w_{ij}(t+1) = \eta\delta f(\zeta) + \alpha\Delta w_{ij}(t) \qquad (6)$$

where η is the learning rate, α is the momentum term which determines how much is remembered about the previous iteration, and δ is the filtered error signal.

Similarly, the internal thresholds θ_j are corrected adaptively in the threshold space. The partial derivatives $-\partial E/\partial w$ and $-\partial E/\partial\theta$ are computed by propagating error signals from the output layer back to the lower layers through the net, which motivates the name "back-propagation".

Parallel implementation of the back-propagation model for Chinese character recognition on hypercubes based on a character decomposition technique using bitmap masks has been discussed by [11.] Our current implementation of the path planner is based on a distribution of the set of training patterns over the number of processors of a hypercube. Essentially, each processor of an allocated hypercube or Transputer-based concurrent processor is responsible for only a small subset of the set of training samples. Using Parasoft EXPRESS as the communication software the same code runs on NCUBE-1, iPSC-2, and on Meiko Computing Surface which is a Transputer-based system.

The training set we used consists of 184 patterns, which contains knowledge of 184 scenarios of human-supervised optimal and collision-free moves in binary terrain navigation. The choice of using 184 patterns is arbitrary. Our first goal is to teach the navigator enough basic knowledge upon which it can generalize, not just memorize, to cope with most situations.

Learning Histories of A Back-Propagation Path Planner

Besides the issue of pattern representation, parameter tuning is a major concern for the back-propagation model to converge fast, or converge at all. Convergence of the model depends on the initial configuration of the network, the choice of the learning rate η, and the momentum term α.

The learning histories for four different initial configurations with fixed learning rate $\eta = 0.10$, and momentum $\alpha = 0.80$ are shown in Fig. 3. Average error is defined as the average of the total quadratic error per pattern per output neuron. The back-propagation path planner converged for these four cases.

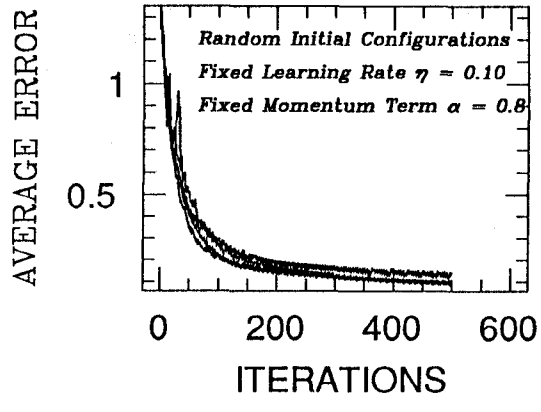

Figure 3: The learning histories of a back-propagation path planner for four different initial configurations.

To study the effect of the momentum term α on the convergence of the model, we used a fixed initial configuration and set $\eta = 0.10$. Figure 4 shows the learning histories for $\alpha = 0.2$, 0.4, 0.6, and 0.8. The behavior of the model in its learning phase were very similar for the four different values of α. All four cases converged roughly at the same rate because they had the same learning rate.

More dramatic effects were observed, as expected, for the cases of using different learning rates η, and fixing the initial configuration and the value of α. The learning histories for seven different learning rates are displayed in Fig. 5.

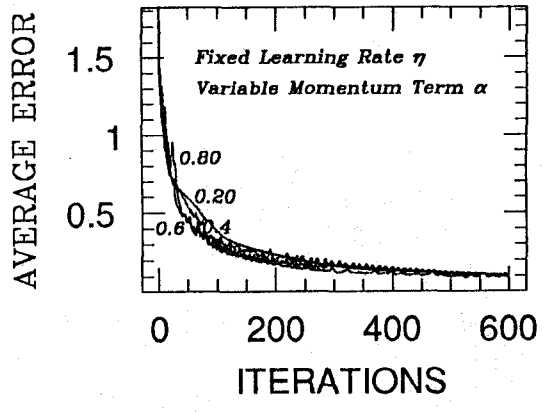

Figure 4: The learning histories of a back-propagation path planner using four different values for the momentum term.

In general, the bigger the learning rate the faster the convergence. However, when the learning rate is set to a value that is "too big", it leads to big oscillations and instability. Small learning rates usually lead to smooth but slow learning.

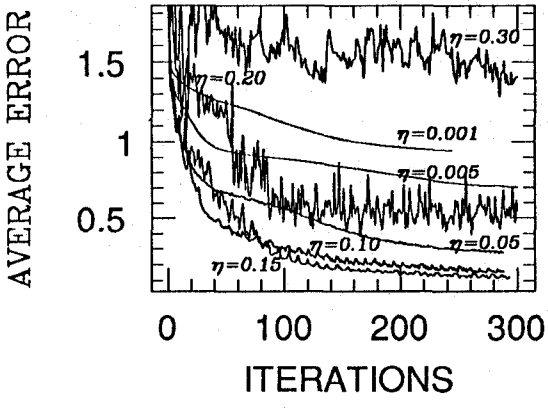

Figure 5: The learning histories of a back-propagation path planner for seven different learning rates.

Performance of A Trained Back-Propagation Path Planner

We tested the performance of a trained back-propagation neural path planner by submitting to it unlearned scenarios in the form of 47 x 24 as well as 105 x 53 discretized map. Several instances which include random and structured obstacles are shown in Fig. 6 to 10.

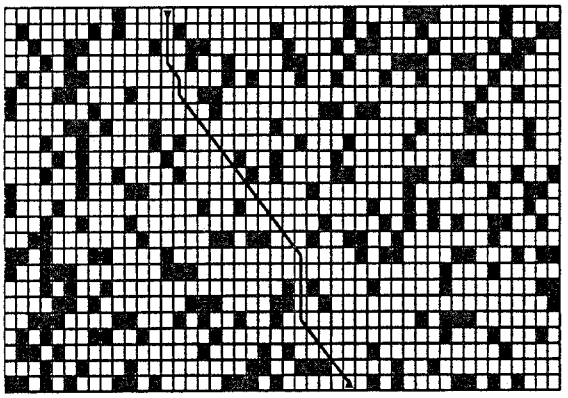

Figure 6: A 47 x 24 binary terrain with randomly distributed obstacles. Triangle indicates starting location, and an inverted triangle indicates target position. An optimal collision-free path was planned.

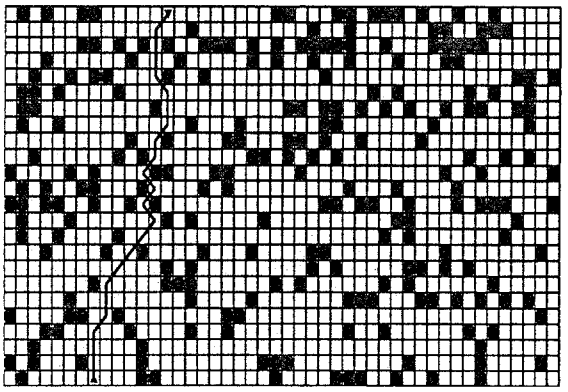

Figure 7: A 47 x 24 binary terrain with randomly distributed obstacles. An optimal collision-free path was planned.

Since an optimal path in this study is one which minimizes the number of collision-free moves needed to get from a source to a target position within the problem domain, the paths displayed in Fig. 6, 7, 9, and 10 are optimal. However, the planned path is near-optimal in Fig. 8.

Performance of Parallel Implementation

We used up to 64-node NCUBE-1, iPSC-2, and 16-node Meiko for our simulations. A training set of 184 patterns becomes a small problem for the case of 64 processors because each processor is then responsible for performing computations for at most 3 patterns. Figure 11 shows the timing result for running one iteration of the back-propagation path planner. The reported time is normalized to that needed to run one iteration of the same planner on a 20 MHz SUN4/60 SPARCstation 1. For the one processor case, a Meiko Transputer node was the fastest, achieving a performance close to that of a SUN4/60. The efficiencies of the same program on the three different concurrent processors are shown in Fig. 12. Although the efficiency for the simulations performed on an NCUBE-1 seems to be better than on an iPSC-2 or a Meiko Transputer systems, this result should be taken with care. Even though exactly the same EXPRESS program was used for simulations, there were differences in the hard-wired configuration of the three computer systems, and in the implementation of EXPRESS.

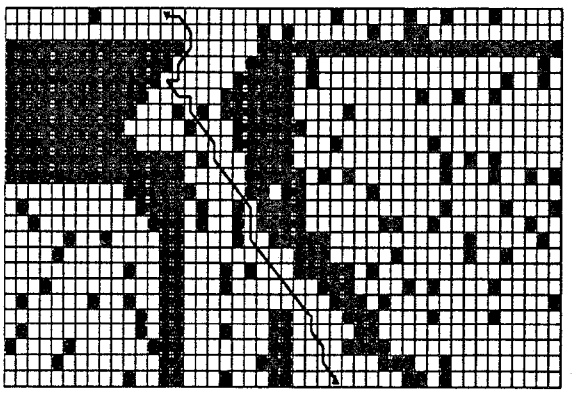

Figure 8: A 47 x 24 binary terrain with structured obstacles. The planned path is near-optimal.

There is one simple explanation for the poor efficiency of the simulations on a Meiko Computing Surface. EXPRESS communications are best suited for hypercube connectivity concurrent processors. At the time of the simulations, the Transputer system was hard-wired as a 2-d torus instead of a hypercube; thus, resulting in poor performance.

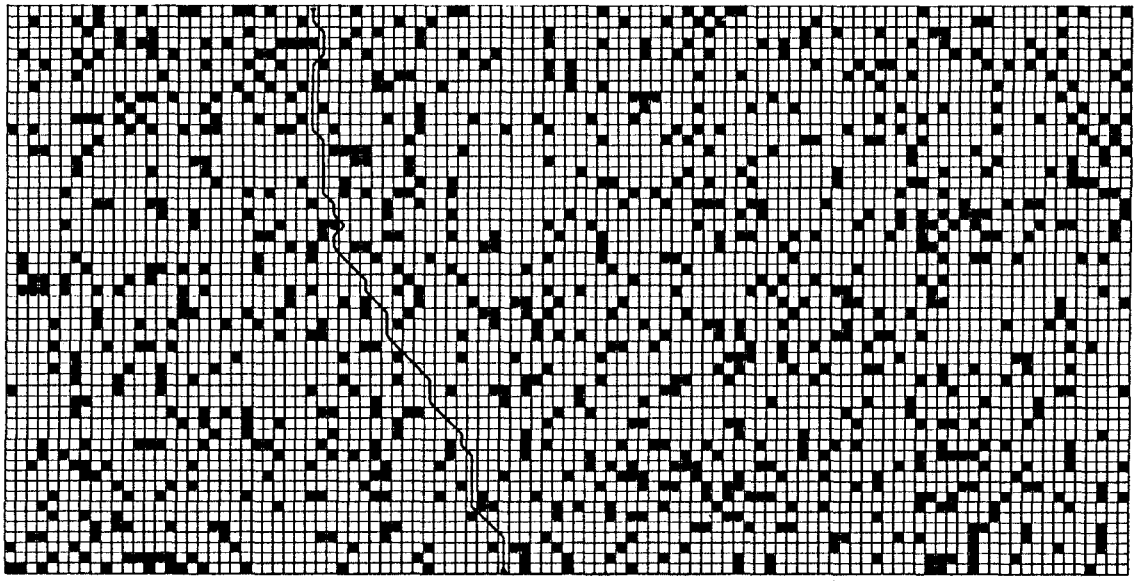

Figure 9: A 105 x 54 binary terrain with randomly distributed obstacles. An optimal collision-free path was planned.

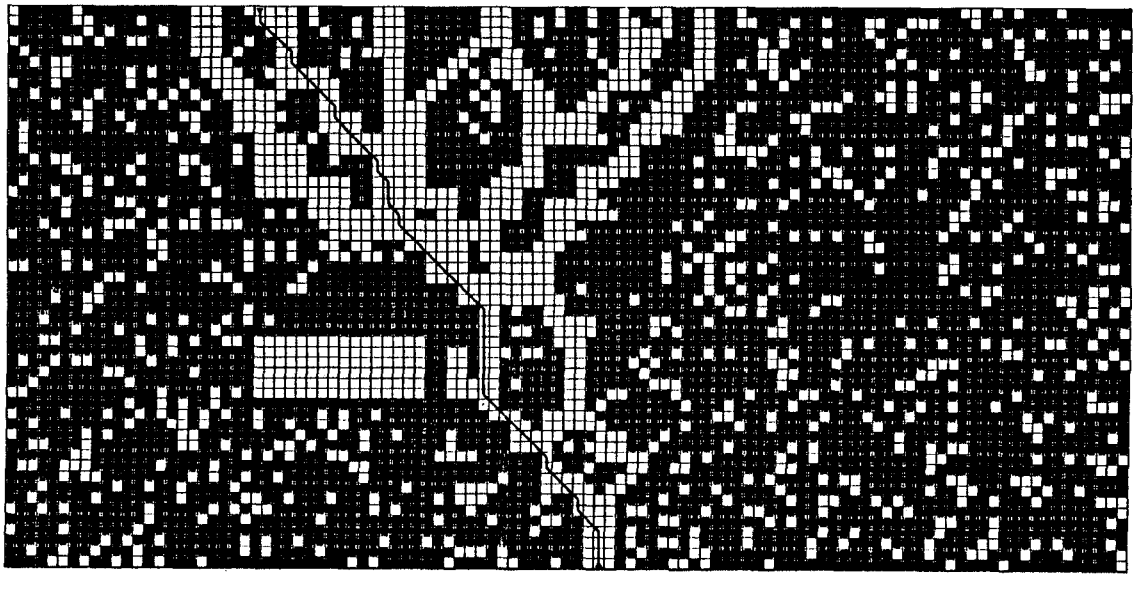

Figure 10: A 105 x 54 binary terrain with structured obstacles. An optimal collision-free path was planned.

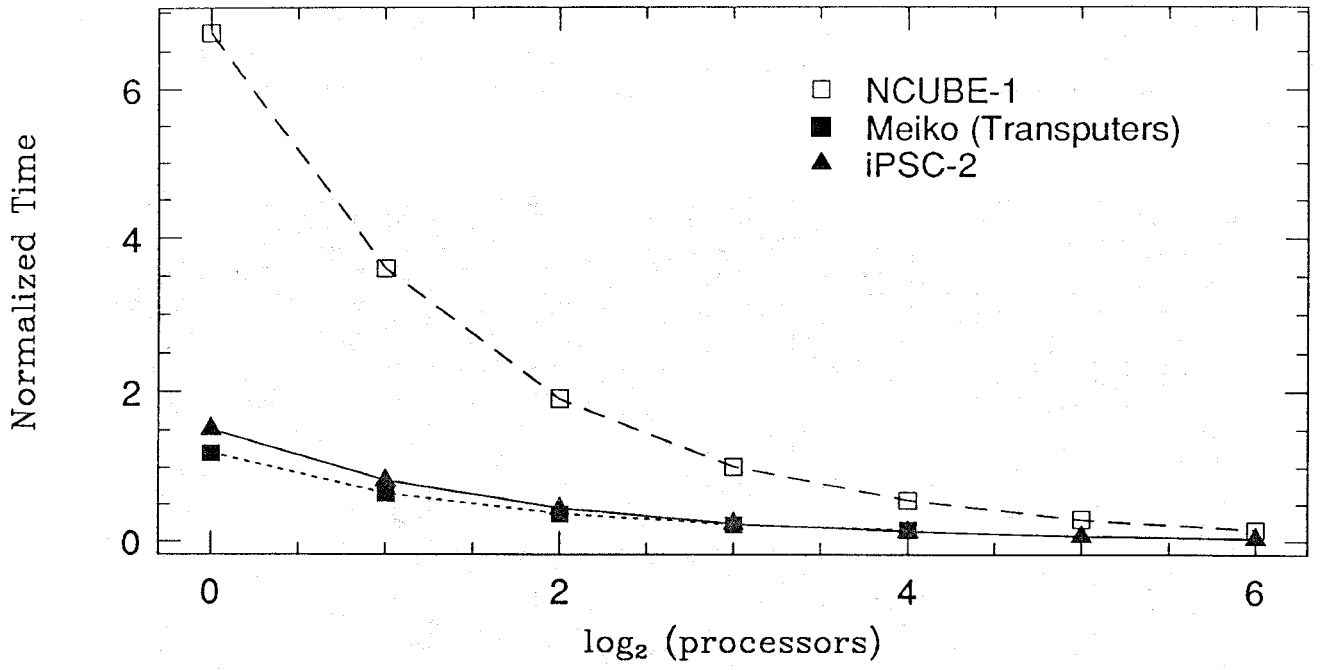

Figure 11: Timing result normalized to the time required to run one epoch on a 20 MHz SUN4/60 SPARCstation 1.

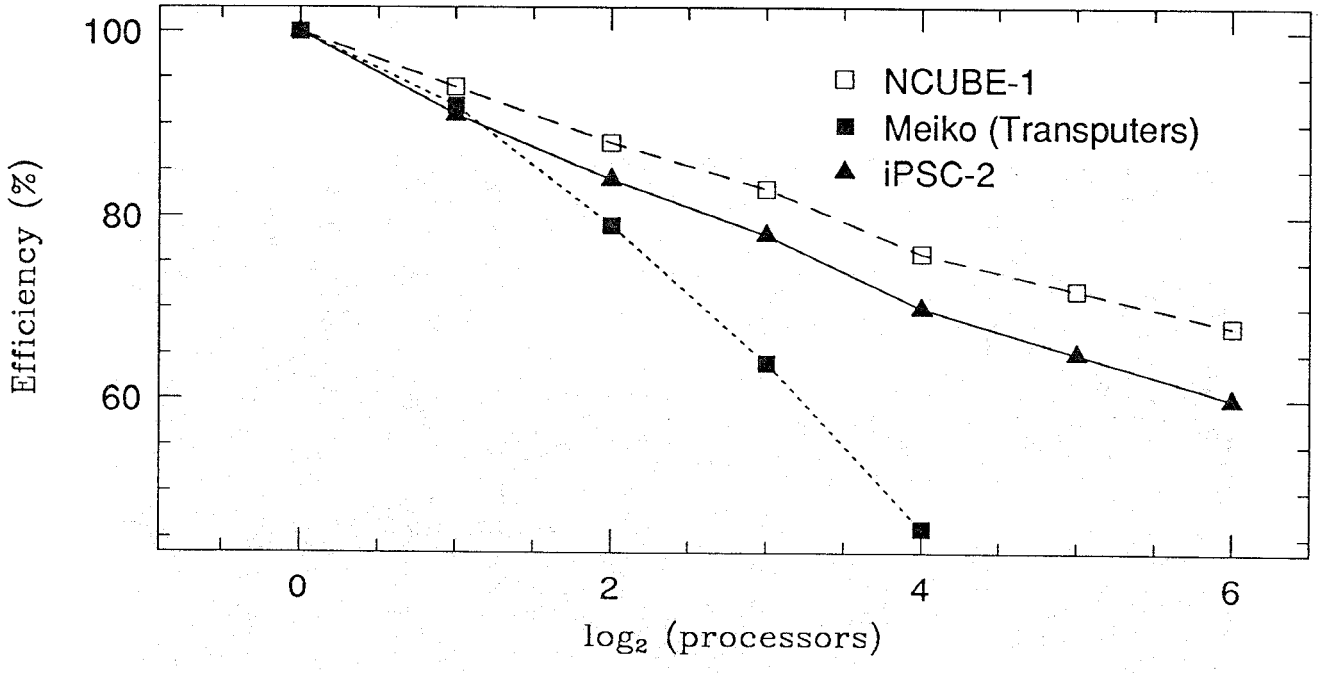

Figure 12: Efficiency plotted as a function of the dimension of a hypercube.

EXPRESS was optimally implemented on the NCUBE-1. As for the iPSC-2, EXPRESS was implemented on top of the native operating system (NX). This extra layer in between EXPRESS and the hardware is expected to incur inefficiency.

Conclusions

Simulation results indicate that a trained back-propagation path planner possesses satisfactory capability of planning near-optimal, collision-free paths in binary terrains with random or structured obstacles. The multi-scale mapping scheme does not only reduce the size of the computational domain and encode sufficient information to carry out the planning task, but also ensures applicability of the trained network on a wide range of problem sizes.

The advantages of this new approach of transforming a path planning problem to one in pattern classification by neural networks are:

- External homing strategy is not required.
- No explicit heuristic is used for shortest path.
- No need to decompose the problem domain into configuration space and free space.

The homing strategy, and the notions of optimality and obstacles avoidance are all encapsulated into the training patterns as task-specific knowledge from a human teacher.

Acknowledgement

This study is based on research work supported by the Joint Tactical Fusion Program Manager.

References

[1] Schwartz, J.T. and Sharir, M., "A Survey of Motion Planning and Related Geometric Algorithms," *Artificial Intelligence* **37**, 157-169 (1988.)

[2] Simic, P., "Statistical Mechanics as the Underlying Theory of 'Elastic' and 'Neural' Optimizers," *NETWORK: Computation in Neural Systems* **1**, 1-15 (1990), Technical Report CALT-68-1556, C^{3}P-787, California Institute of Technology, May 1989.

[3] Geoffrey Fox, Wojtek Furmanski, Alex Ho, Jeff Koller, Petar Simic, and Issac Wong, "Neural Networks and Dynamic Complex Systems", contribution to the 1989 SCS Eastern Conference, Tampa, Florida, (March 1989,) Technical Report C^{3}P-695, California Institute of Technology, 1988.

[4] Hopfield, J. J. and Tank, D.,"Neural Computation of Decisions in Optimization Problems," *Biol. Cybernetics* **52**, 141-152 (1985.)

[5] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., "Optimization by Simulated Annealing," *Science* **220**, 671 (1983.)

[6] Fox, Geoffrey C. and Gurewitz, Eitan and Wong, Yiu-fai, "A Neural Network Approach to Multi-vehicle Navigation", contribution to the 1989 SPIE Conference, Philadelphia, Pennsylvania, (Nov. 1989,) Technical Report C^{3}P-833, California Institute of Technology, 1989.

[7] Wong, Issac and Fox, Geoffrey C., "Use of neural networks for path planning," Technical Report C^{3}P-784, California Institute of Technology, May 1989.

[8] Ho, Alex W., "A Back-propagation Navigation Controller for Land and Space Vehicles," Technical Report C^{3}P-735, California Institute of Technology, April 1989.

[9] Ho, Alex W. and Fox, Geoffrey C, "Parallel Neural Net Planner on Hypercube and Transputer," in **Parallel Processing in Neural Systems and Computers,** eds. R. Eckmiller et. al., Elsevier, **421-426,** (1990.) Technical Report C^{3}P-848, California Institute of Technology, 1989.

[10] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representation by Error Propagation" in D.E. Rumelhart & J.L. McClelland (Eds.), "Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations." MIT Press, 318-364 (1986.)

[11] Ho, Alex W. and Furmanski, W., "Pattern Recognition using Neural Networks in Hypercubes," in the Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications, **Vol. 2**, (ed.) Geoffrey C. Fox, ACM Press, New York, 1011-1021 (1988), Technical Report C^{3}P-528, California Institute of Technology, 1988.